UNIVERSITÀ DEGLI STUDI DI UDINE

Analisi e realizzazione di codice Minizinc e ASP per la creazione di squadre di pallacanestro

DITTARO FEDERICO 154538 dittaro.federico@spes.uniud.it

Progetto Automated Reasoning anno accademico 2023/2024
Corso di laurea in Artificial Intelligence and Cybersecurity

1. DEFINIZIONE DEL PROBLEMA

Il problema assegnato prevede la creazione di squadre di pallacanestro a partire da una lista di giocatori $p_1 \ldots p_n$ in cui n è, per comodità, un multiplo di 5. Ogni giocatore assegna un punteggio da 1 a 5 alle possibili cinque posizioni (1 rappresenta il ruolo di point guard, 2 il ruolo di shooting guard, 3 il ruolo di small forward, 4 il ruolo di power forward e 5 il ruolo di centro) in cui 1 rappresenta il voto minimo (non vuole giocare in quel ruolo) e 5 il voto massimo (è il suo ruolo preferito). È inoltre presente una lista di "hard constraint" chiamata *different_team(p_i, p_j)* in cui viene specificata la coppia di giocatori che non possono stare nella stessa squadra.
L'input del problema è quindi fornito dalle seguenti informazioni:

- Lista dei giocatori presenti;
- Altezza in centimetri dei giocatori;
- Punteggi assegnati da ogni giocatore ad ogni ruolo;
- Hard constraint.

L'obiettivo è quello di creare k squadre (dove $k = n / 5$) cercando di massimizzare le preferenze personali in modo tale che siano soddisfatti i seguenti vincoli:

1. Vincolo del ruolo: ogni squadra ha esattamente un giocatore per ruolo;
2. Vincole sulle altezze: sia t_l la somma delle altezze dei giocatori della squadra l, per ogni i, j ϵ 1..k $t_i - t_j < 20$;
3. Hard constraint: due giocatori inseriti nel predicato different_team non possono far parte delle stessa squadra;
4. Calcolo della penalità: la penalità viene calcolata per ogni giocatore la cui posizione in squadra non è la sua preferita (ovvero quella a cui ha assegnato il punteggio 5), se ad esempio il ruolo preferito di un giocatore è il centro, ma viene assegnato ad un'altra posizione, allora la penalità sarà 5 - X dove X rappresenta il punteggio assegnato alla posizione corrispondente. L'obiettivo è quello di minimizzare la penalità.

2. MODELLO IN MINIZINC

In questa sezione verranno illustrate le scelte effettuate per rappresentare l'input, l'output e la modalità con cui sono stati stanziati i vari vincoli necessari per soddisfare il problema originale.

2.1 DEFINIZIONE DELL'INPUT

Ogni file "x-players-instance.txt" contiene un elenco di x giocatori tutti diversi tra loro e inoltre definisce le seguenti variabili:

- int: maxt; questo valore permette di definire il numero di squadre (corrisponde al valore *k* definito nella prima sezione), il suo valore cambia in base al numero dei giocatori presenti;
- int: maxp; questo valore rappresenta il numero di giocatori (multiplo di 5). Per comodità di lettura del risultato è stato definito l'array "playersNames" in cui viene associato un nome ad ogni giocatore.

Per la definizione delle altezze è stato creato un array "playersHeights" di dimensione corrispondente al numero di giocatori.
Per quanto riguarda la definizione dei punteggi è stata definita la matrice "playersScores" di dimensione [maxp x 5] che permette quindi di rappresentare le righe come i giocatori e le colonne come le posizioni. In questo modo ad ogni cella corrisponde il voto di un determinato giocatore per una determinata posizione.
L'ultimo elemento da definire è l'hard constraint, in questo caso si è utilizzata una matrice di dimensione [maxp x maxp] in cui ad ogni cella viene assegnato un valore 0 oppure 1 (0 rappresenta il fatto che i due giocatori possono stare nella stessa squadra, 1 il fatto che devono stare in squadre diverse).

```
% Definizione dei punteggi
array[1..maxp, 1..5] of int: playersScores = array2d(1..maxp, 1..5, [
    4, 4, 3, 2, 1,  % federico
    4, 5, 3, 2, 1,  % riccardo
    3, 4, 5, 2, 1,  % andrea
    1, 2, 3, 5, 4,  % alessandro
    1, 2, 3, 4, 5,  % stefano
    5, 4, 3, 2, 1,  % leonardo
    4, 5, 3, 2, 1,  % carlo
    3, 4, 5, 2, 1,  % matteo
    1, 2, 3, 5, 4,  % mattia
    1, 2, 3, 4, 4  % kevin
]);

% Definizione different_team (1 non possono stare nella stessa squadra, 0 si)
% In questo caso (Federico, Riccardo) e (Andrea, Matteo)
array[1..maxp, 1..maxp] of 0..1: different_team = array2d(1..maxp, 1..maxp, [
    0, 1, 0, 0, 0, 0, 0, 0, 0, 0,  % federico
    1, 0, 0, 0, 0, 0, 0, 0, 0, 0,  % riccardo
    0, 0, 0, 0, 0, 0, 0, 1, 0, 0,  % andrea
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  % alessandro
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  % stefano
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  % leonardo
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  % carlo
    0, 0, 1, 0, 0, 0, 0, 0, 0, 0,  % matteo
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  % mattia
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0  % kevin
]);
```

Esempio di popolamento delle matrici "playersScores" e "different_team".

2.2 DEFINIZIONE DEI VINCOLI

Tutti i vincoli spiegati in questa sezione fanno riferimento agli array unidimensionali "team_created" e "players_roles". Il primo ha una dimensione corrispondente al numero di giocatori e l'idea è quella di assegnare ad ogni cella un valore x tale che $1 \leq x \leq maxt$. In questo modo ogni giocatore verrà assegnato ad una sola squadra. Il secondo, di dimensione uguale, viene utilizzato per memorizzare le posizioni assegnate ai giocatori durante la computazione e contiene valori da 1 a 5. Questo array viene utilizzato per rispettare il vincolo numero 1 definito nel primo capitolo.

Sono stati definiti i seguenti vincoli:

- **Vincolo sul numero di giocatori**: ogni squadra deve avere esattamente 5 giocatori e ogni giocatore deve appartenere ad una sola squadra. L'obiettivo è stato raggiunto sfruttando la funzione "*count"* la cui sintassi è *count(array, variabile, intero).*
  Attraverso un ciclo "forall" che attraversa tutti i valori da 1 a maxt (quindi tutte le squadre) vengono contate il numero di occorrenze di quella determinata squadra all'interno dell'array. Imponendo questo valore a 5 vengono create squadre di esattamente 5 giocatori tutti diversi fra loro;

```
% Ogni squadra ha esattamente 5 giocatori
constraint forall(t in 1..maxt) (
    count(team_created, t, 5)
);
```

- **Vincolo sui ruoli**: ogni squadra deve avere un giocatore per ruolo. Il vincolo viene soddisfatto attraverso due cicli innestati, il primo permette di scorrere l'insieme delle squadre mentre il secondo permette di specificare l'assenza di due giocatori con lo stesso ruolo all'interno della stessa squadra;

```
% Ogni squadra ha esattamente un giocatore per ruolo
constraint forall(t in 1..maxt) (
    forall(i, j in 1..maxp where team_created[i] = t /\ team_created[j] = t /\ i != j) (
     players_roles[i] != players_roles[j]
  )
);
```

- **Vincolo sulle altezze**: sia t_l la somma delle altezze dei giocatori della squadra *l*, per ogni i, j ϵ 1..k $t_i - t_j < 20$. Viene utilizzato un ciclo per attraversare tutte le possibili coppie di squadre utilizzando due variabili temporanee contenenti la somma delle altezze della rispettiva squadra. La loro differenza in valore assoluto deve essere inferiore a 20.

```
% Vincolo sulle altezze dei giocatori
constraint forall(t, k in 1..maxt where t != k) (
          let{
            var int: sum1 = sum(i in 1..maxp where team_created[i] = t)(playersHeights[i]),
            var int: sum2 = sum(j in 1..maxp where team_created[j] = k)(playersHeights[j])
          } in
          abs(sum1-sum2) < 20
);
```

- **Verifica del vincolo different_team**: non possono stare nella stessa squadra due giocatori per cui è specificato il vincolo di different_team. Per come è stato definito il vincolo, è sufficiente verificare che due giocatori non abbiano impostato il valore 1 nella rispettiva posizione.

```
% Verifica del vincolo different_team
constraint forall(t in 1..maxt) (
      forall(i, j in 1..maxp where team_created[i] = t /\ team_created[j] = t /\ i != j) (
          different_team[i, j] != 1 /\ different_team[j, i] != 1
      )
);
```

- **Calcolo della penalità**: per ottenere la penalità è necessario ricavare la posizione assegnata ed il corrispondente voto per ogni giocatore. Il calcolo avviene sottraendo dal massimo punteggio possibile (dato da 5*n dove n rappresenta il numero di squadre) la somma dei voti. Imponendo di minimizzare questo risultato, le squadre create rispecchieranno il più possibile le scelte personali di ogni giocatore.

```
% Calcolo della penalità
var int: sum_votes = sum(i in 1..maxp, t in 1..maxt where team_created[i] = t)(
    playersScores[i, players_roles[i]]
);

var int: penalty = 25*maxt - sum_votes;

solve minimize penalty;
```

2.3 OUTPUT ED ESEMPI DI ESECUZIONE

L'output è composto dall'array "team_created", dall'array "players_roles" e della minima penalità ottenuta. Utilizzando ad esempio come istanza di input il file "10-players-instance.txt" si ottiene il seguente risultato:

```
Squadra: [2, 1, 1, 2, 1, 1, 2, 2, 1, 2]
Ruoli:   [1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
Penalità: 0
----------
```

Le squadre create sono quindi le seguenti (si può facilmente verificare che il risultato soddisfa tutti i vincoli):

- SQUADRA 1: Riccardo, Andrea, Stefano, Leonardo, Mattia
- SQUADRA 2: Federico, Alessandro, Carlo, Matteo, Kevin

In questo caso la penalità è 0, in quanto l'input consente di creare squadre che rispecchiano esattamente tutte le preferenze di tutti i giocatori. I file "x-players-instance-penalty.txt" permettono di creare delle squadre con delle inevitabili penalità.

3. MODELLO IN CLINGO

In questa sezione verranno illustrate le scelte effettuate per rappresentare l'input, l'output e la modalità con cui sono stati stanziati i vari vincoli necessari per soddisfare il problema originale.

3.1 DEFINIZIONE DELL'INPUT

I file "x-players-instance.txt" contengono l'input, definito dai seguenti predicati:

- teams(1..x): rappresenta il numero di squadre. Il valore x varia in base al numero di giocatori;
- position(1..5): definisce il numero di posizioni della pallacanestro (1 rappresenta il ruolo di point guard, 2 il ruolo di shooting guard, 3 il ruolo di small forward, 4 il ruolo di power forward e 5 il ruolo di centro);
- player(nome): definisce i giocatori assegnando un nome. Il numero totale varia in base al numero di squadre da creare
- height(player, valore): assegna ad ogni player la rispettiva altezza;
- punteggio(player, position, voto): permette di assegnare ad ogni player un voto per ogni posizione. Ogni giocatore ha cinque istanze di questo tipo;
- different_team(player, player): definisce i giocatori che non possono stare nella stessa squadra.

```
player(federico).
height(federico, 178).
punteggio(federico, 1, 5).
punteggio(federico, 2, 4).
punteggio(federico, 3, 3).
punteggio(federico, 4, 2).
punteggio(federico, 5, 1).
```

Esempio di inizializzazione del giocatore "federico"

3.2 DEFINIZIONE DEI VINCOLI

In questa sezione viene introdotto il predicato “team_created(T, G, P)” dove T rappresenta un team, G un giocatore e P la rispettiva posizione all’interno della squadra. Questo viene creato all’interno del primo vincolo riportato qui sotto.

- **Vincolo sul numero di giocatori**: ogni squadra deve avere esattamente 5 giocatori e ogni giocatore deve appartenere ad una sola squadra. Il vincolo è stato ottenuto attraverso i predicati riportati nell’immagine: il primo permette di contare il numero di squadre e posizioni, imponendo che per ogni giocatore vengano assegnati esattamente una sola squadra ed un solo ruolo, mentre il secondo permette di contare il numero di giocatori e posizioni, imponendo che per ogni squadra vengano assegnati esattamente cinque giocatori e cinque ruoli;

```
% Vincoli che creano squadre di esattamente 5 giocatori tutti diversi
1{team_created(T,G,P) : team(T), position(P)}1 :- player(G).
5{team_created(T,G,P) : player(G), position(P)}5 :- team(T).
```

- **Vincolo sui ruoli**: ogni squadra deve avere un giocatore per ruolo. Questo viene rispettato imponendo l’assenza di due giocatori (diversi fra loro) appartenenti alla stessa squadra e con lo stesso ruolo;

```
% Ogni squadra ha un giocatore per ruolo
:- team_created(T, G1, P), team_created(T, G2, P), G1 != G2.
```

- **Vincolo sulle altezze**: sia t_i la somma delle altezze dei giocatori della squadra *i*, per ogni i, j ϵ 1..k $t_i - t_j < 20$. È stata creata una funzione “sum_heights(T, SUM)” dove T è una variabile utilizzata per attraversare tutte le squadre e SUM è la variabile contenente la somma delle altezze dei rispettivi giocatori. Il vincolo viene verificato imponendo che non debbano esistere due squadre la cui differenza di altezze sia superiore a 20 centimetri;

```
% Funzione per calcolare la somma delle altezze
sum_heights(T, SUM) :- team(T), SUM = #sum{H : team_created(T, G, _), player(G), height(G,H)}.

% Vincolo sulle altezze dei giocatori
:- team(T1), team(T2), T1 != T2, sum_heights(T1, X), sum_heights(T2, Y), X-Y > 19.
```

- **Verifica del vincolo different_team**: non possono stare nella stessa squadra due giocatori per cui è specificato il vincolo di different_team. Questo viene ottenuto semplicemente verificando che una coppia di due giocatori appartenenti alla stessa squadra non faccia contemporaneamente parte dell’hard constraint different_team;

```
% Verifica del vincolo different_team
:- team_created(T, G1, _), team_created(T, G2, _), different_team(G1, G2), G1 != G2.
```

- **Calcolo della penalità**: per ottenere la penalità è necessario ricavare la posizione assegnata ed il corrispondente voto per ogni player. Viene inizialmente calcolata la penalità per ogni giocatore ricavando la posizione in cui è stato assegnato, successivamente tutti i risultati vengono sommati fra di loro. Imponendo di minimizzare questo valore, le squadre create rispetteranno il più possibile le scelte personali dei singoli giocatori.

```
% Calcolo della penalità
player_penalty(T, G, Pen) :- team(T), team_created(T, G, R), punteggio(G, R, S), Pen = 5-S.

penalty(P) :- P = #sum{Pen, G: player_penalty(_, G, Pen)}.

#minimize {P:penalty(P)}.
```

3.3 OUTPUT ED ESEMPI DI ESECUZIONE

L'output è composto dai predicati "team_created" che permette di visualizzare per ogni giocatore la squadra ed il ruolo assegnati, "sum_heights" che mostra la somma delle altezze per ogni squadra e la variabile penalty che mostra la minima penalità trovata.
Utilizzando ad esempio il file "15-players-instance.txt" si ottiene il seguente risultato:

```
team_created(1,federico,1) team_created(3,riccardo,2) team_created(1,andrea,3)
team_created(2,alessandro,4) team_created(3,stefano,5) team_created(2,leonardo,1)
team_created(1,carlo,2) team_created(3,matteo,3) team_created(1,mattia,4)
team_created(1,kevin,5) team_created(2,simone,2) team_created(2,gianni,3)
team_created(3,daniele,4) team_created(2,gregorio,5) team_created(3,ruben,1)
penalty(0) sum_heights(1,926) sum_heights(2,916) sum_heights(3,920)
Optimization: 0
OPTIMUM FOUND
```

Le squadre create sono quindi le seguenti:

- SQUADRA 1 (altezza: 926 cm): Federico, Andrea, Carlo, Mattia, Kevin;
- SQUADRA 2 (altezza: 916 cm): Alessandro, Leonardo, Simone, Gianni, Gregorio;
- SQUADRA 3 (altezza: 920 cm): Riccardo, Stefano, Matteo, Daniele; Ruben.

In questo caso la penalità è 0, in quanto l'input consente di creare squadre che rispecchiano esattamente tutte le preferenze di tutti i giocatori. I file "x-players-instance-penalty.txt" permettono di creare delle squadre con delle inevitabili penalità.

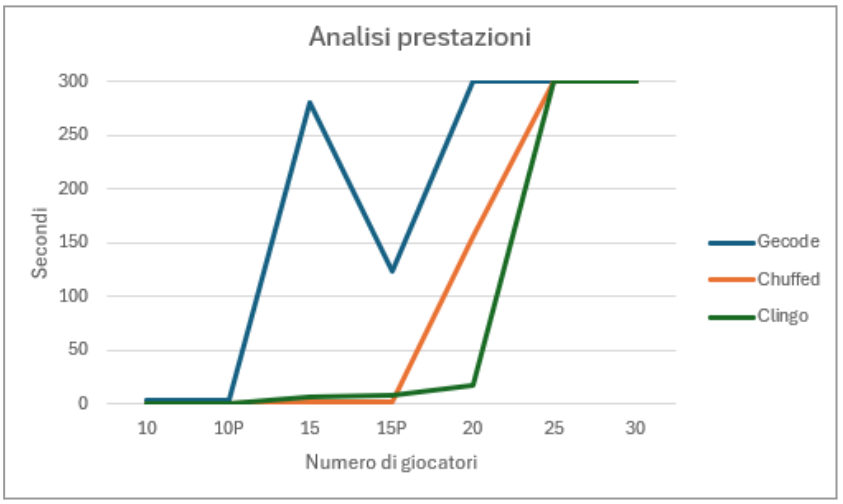
4. RISULTATI

In questa sezione verranno presentati attraverso grafici e tabelle i principali risulati ottenuti dall'analisi delle prestazioni nel caso dei due linguaggi.

La seguente tabella riassume il tempo necessario per formare le squadre con l'utilizzo di Minizinc e Clingo. Per quanto riguarda il primo sono stati utilizzati due solver: Gecode 6.3.0 e Chuffed 0.12.1.
Nella tabella il numero di giocatori con o senza penalità sono riportati nelle righe, mentre il solver utilizzato è riportato nella colonna. Il TIMEOUT utilizzato nei test è di cinque minuti. La penalità riportata è quella calcolata nei 5 minuti.

	Gecode 6.3.0	**Chuffed 0.12.1**	**Clingo**
10 giocatori senza penalità	3s 905msec	355msec	219msec
10 giocatori con penalità	3s 921msec (penalità 2)	321msec (penalità 2)	193msec (penalità 2)
15 giocatori senza penalità	4m 40s	1s 65msec	7s 402msec
15 giocatori con penalità	1m 58s (penalità 2)	1s 7msec (penalità 2)	8s 77msec (penalità 2)
20 giocatori senza penalità	TIMEOUT (penalità 17)	2m 36s	18s 186msec
25 giocatori senza penalità	TIMEOUT	TIMEOUT (penalità 5)	TIMEOUT (penalità 17)
30 giocatori senza penalità	TIMEOUT	TIMEOUT	TIMEOUT

Si può osservare come Gecode sia il solver più lento fra i tre proposti: Nei 5 minuti è infatti in grado di creare solamente tre squadre (15 giocatori), nel caso di 20 fornisce un risultato non ottimale, mentre nel caso di 25 e 30 non fornisce alcuna combinazione.
Per quanto riguarda invece Chuffed e Clingo entrambi riescono a creare fino a 4 squadre ottimali (20 giocatori), con 25 forniscono un risultato non ottimale mentre con 30 non producono nessuna combinazione.
L'aumento della complessità può essere osservata più facilmente attraverso il seguente grafico (P rappresenta le istanze per cui è prevista una penalità):



5. OTTIMIZZAZIONI

Per quanto riguarda Minizinc, per ridurre lo spazio di ricerca e di conseguenza il tempo di computazione necessario, è stato specificato un vincolo per inizializzare l'appartenenza di un determinato giocatore ad una determinata squadra.
Vincolando ad esempio il primo giocatore (Federico) alla squadra 2, i tempi di risoluzione si abbassano. Considerando il solver Chuffed 0.12.1, la riduzione di tempo ottenuta è la seguente:

	Chuffed iniziale	**Chuffed con inizializzazione**
10 giocatori senza penalità	355msec	335msec
10 giocatori con penalità	321msec (penalità 2)	311msec (penalità 2)
15 giocatori senza penalità	1s 65msec	928msec
15 giocatori con penalità	1s 7msec (penalità 2)	740msec (penalità 2)
20 giocatori senza penalità	2m 36s	41s 725msec
25 giocatori senza penalità	TIMEOUT (penalità 5)	TIMEOUT (penalità 10)
30 giocatori senza penalità	TIMEOUT	TIMEOUT

La stessa idea può essere applicata al codice ASP.